# Introduction

In the world of Java programming, RMI stands for Remote Method Invocation which is referred as an important application programming interface which allows one computer or JVM which stands from Java Virtual Machine to connect with objects on another JVM via a distributed network. RMI serves as a conduit for creating distributed Java applications through simplicity of method calls. RMI is also considered to be matched effectively with the concept of a Java Remote Procedure Call (RPC) but extends its capabilities by allowing the delivery of one or more objects alongside request. This complex system mainly uses two basic components to facilitate smooth remote interactions between application: the ’stub’ and the ’skeleton’ (Awati, 2019).

## Background and Context of RMI application development project

The RMI application development project began in broader concept of distributed computing, an approach that is used when expanding program capabilities by distributing components over multiple machines. The incentive for distributed computing rose from the need to handle issues such as fault tolerance, scaling and geographical spread. The application in this system may continue to work seamlessly even in the presence of faults, ensuring uninterrupted service across varied components.

Java perfectly aligns with RMI, making it a popular alternative for designing distributed systems as the synergy between java and RMI, multiple components of the application can communicate continuously, with servers and clients engaging coherently. RMI takes advantage of Java objects' ability to communicate openly over a network, which contributes to the project's robust and versatile architecture.

RMI can be used for variety of purposes such as developing client-server applications, the building of distributed databases, and the setting up of remote administration and control systems. The RMI technique is based on an object-oriented paradigm, which allows for seamless functionality in a distributed context. This object-oriented paradigm, which forms the foundation of Remote Machine Interface (RMI), allows objects dispersed across a network to access their methods. This not only simplifies the design and conceptualization of distributed systems, but it also improves the application's overall efficiency and coherence.

The security of any distributed systems is crucial and RMI addresses such concerns by executing comprehensive security standards. Specific use cases must be defined within the context of the RMI project to tailor the development process to the CKF Group Malaysia's particular goals and objectives. Given the project's distributed nature, maintaining safe interactions between components is critical. To protect RMI interactions and the overall system's integrity, the project will need to investigate and implement best practices.

## RMI Framework and its Role in Distributed Systems

Remote Method Invocation (RMI) is a key Java technology for the construction of distributed systems, allowing for seamless communication between Java objects. This technology allows objects to interact with one another and invoke methods remotely, even if they are on different Virtual Machines or environments. The essence of RMI is its ability to remotely activate methods or objects, allowing dispersed systems to run more efficiently. RMI becomes the backbone of programs that demand collaboration between diverse components spread across different computer systems by facilitating communication among objects across different machines.

RMI's critical role in distributed systems can be summarized as follows:

* Method Invocation and Object Communication: RMI acts as a conduit for objects to communicate with one another and invoke methods on remote machines. This functionality is important in instances when application components run on separate machines.
* Bridge between Components and Servers: RMI acts as a bridge to connect components on the server to different machines. This link enables the invocation of methods, allowing for the cohesive and integrated operation of distributed components.
* Handling Network Communication Details: Handling RMI abstracts and managing sophisticated network communication elements, facilitating the design and deployment of distributed systems. This layer hides developers from the complexity of low-level network connection, resulting in a simpler and more efficient development process.
* Rapid create and Deployment: RMI's features make it easier to create and deploy distributed systems. Its architecture allows developers to concentrate on the logic and functionality of the program while RMI handles the underlying communication issues, resulting in a more efficient and rapid development process.

## Cloud Computing and Virtualization

Virtualization is a transformational technology allowing numerous virtual machines (VMs) to process on a single physical device by creating various virtual instances of computing resources. The hypervisor, also known as the Virtual Machine Monitor (VMM), regulates resource allocation, and each VM operates independently with its own operating system. It allows effective utilization of resources through pooling, isolating VMs for increased security, and supports backup and scalability capabilities such as snapshots and cloning.

To abstract and efficiently utilize real infrastructure, virtual instances of computer resources like servers or storage are created. Its significance in distributed systems includes the following:

* Virtualization enables numerous virtual machines (VMs) to run on a single physical server. This consolidation improves overall performance and energy efficiency by optimizing resource consumption in distributed systems.
* Each virtual instance functions independently, allowing applications to be isolated from one another. This improves distributed system security by avoiding potential breaches in one component of the system from affecting others.
* Virtualization permits the dynamic allocation of resources to different components of a distributed system. This adaptability guarantees that applications obtain the resources they require in real-time, allowing them to adjust to changing workloads.
* Managing virtualized resources is frequently simpler than dealing with physical infrastructure. This management simplicity is critical in distributed systems, where coordination and maintenance can become complicated.
* Virtualization enables the construction of isolated testing environments, allowing developers to test distributed programs in controlled environments. This shortens the development time and ensures the robustness of distributed systems.

Cloud computing represents a paradigm shift towards the delivery and consumption of computing services as it provides on-demand services via internet connectivity to a shared pool of adjusting computing resources. Users can access services required, offering a cost-effective service. Cloud computing provides widespread network access, allowing for ubiquitous connectivity from a variety of devices. Because of its resource pooling and quick flexibility, it allows for dynamic scaling to suit shifting workloads. Cloud services are classified as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), providing customers and companies with a variety of possibilities.

Cloud computing provides a scalable and adaptable architecture that enables distributed systems to respond to changing workloads. Its primary significance is found in:

* Scalability: Cloud computing allows for the smooth scaling of resources in response to demand. This ensures that applications in a distributed system can handle higher workloads without losing performance.
* Resource Efficiency: Cloud computing allows resources to be allocated and used on-demand. As a result, resource efficiency improves because distributed systems can dynamically allocate and deallocate resources as needed, optimizing utilization.
* Fault Tolerance: Redundancy and data backup technologies are frequently used by cloud providers. This improves distributed system fault tolerance, maintaining data integrity and system availability even in the face of hardware failures or disruptions.
* Global Accessibility: Cloud services enable worldwide access to distributed systems, allowing users to access apps and data from any location with an internet connection. This matches the scattered nature of modern apps.
* Cost reduction: Cloud computing operates on a pay-as-you-go basis, allowing businesses to pay for just the resources they utilize. This cost optimization is especially useful for distributed systems with varying resource requirements.

# Objective and Goals

* Create a user-friendly and efficient Product Delivery System (PDS) with RMI and Java.
* Enable functionalities like user account registration, authentication, profile management, purchase management with necessary information.
* Create an effective order processing system that provides order confirmation, status updates, and thorough reports to customers.
* Use RMI technology for effective communication among distributed components.
* Quality requirements, such as usability, maintainability, and heterogeneity, should be used to evaluate the implemented application.

# System design

## Overview of RMI application architecture and components

In the RMI architecture, various distributed components contribute to the development of a distributed system that allows seamless communication between clients and servers. These components are important to the overall functionality of the Remote Method Invocation process.

Remote Interface:

The remote interface refers to a base element in the RMI architecture that determines the structure for remote methods that are required to be invoked across different Java Virtual Machines (JVMs). By extending the 'java.rmi.Remote' interface, it establishes a standardized blueprint for method signatures, ensuring consistency among the distributed components.

Remote Objects:

At the core of the RMI architecture are remote objects, Java entities which executes the specified remote interface. These objects are hosted on servers and play a vital role in implementing the actual logic of remote methods. When a client initiates an RMI call, it is these remote objects that handle execution of methods and provide the corresponding response.

Stubs and Skeletons:

Stubs and skeletons act as intermediaries facilitating the communication between clients and servers. On the client side, stubs serve as local proxies to the remote objects, serializing method calls and directing them to the server. On the server side, skeletons receive these calls, invoke the corresponding methods on the actual remote objects, and manage the serialization of results back to the client.

RMI Registry:

The RMI Registry refers to a a central directory registry, providing a mechanism for servers to register their names or URLs. Clients can then validate for specific remote objects using these identifiers, obtaining references that are essential for creating method calls. This centralized registry streamlines the process of object identification and retrieval.

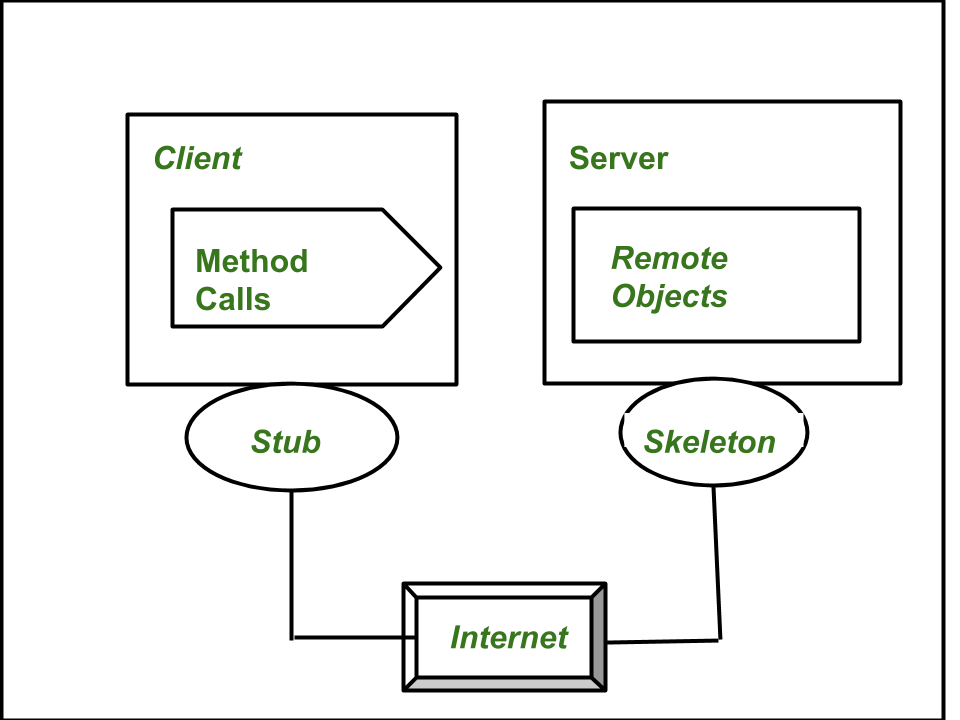
RMI API:

The RMI API, residing in the 'java.rmi' package, is a comprehensive set of interfaces and classes that equip developers with the tools needed to implement and manage RMI applications. It encompasses functionalities related to remote object management, naming, and registry interactions, forming the backbone of RMI application development.

Java Virtual Machine:

The Java Virtual Machine (JVM) is a crucial component responsible for executing Java code on both the client and server sides. It ensures the proper functioning of remote objects and the execution of RMI method calls in a distributed environment. The JVM creates and manages the runtime environment for RMI applications, playing a fundamental role in their seamless operation.

## Interaction of RMI Application Components



i) Stub:

The stub acts as the gateway on the client side, symbolizing the remote object and facilitating outbound requests. When a method is invoked on the stub, a sequence of actions are passed. It initiates a connection to the remote Virtual Machine (JVM), marshals the parameters by writing them to the remote JVM, waits for the response, decodes the return value upon receipt, and ultimately displays the result to the user. The stub plays a critical role in managing the communication flow from the client to the server, ensuring proper data transmission and reception.

ii) Skeleton:

On the server side, the skeleton play a role of a receiver of incoming requests sent towards the server-side object. Once an inbound request is received, the skeleton performs critical tasks. It extracts the arguments of the remote method, invokes the actual method on the true remote object, captures the result, and marshals it back to the caller. The skeleton serves as the intermediary on the server side, orchestrating the execution of remote methods and handling the communication flow between the server and the client.

The interaction between the stub and skeleton is a vital aspect of the RMI architecture. The stub on the client side marshals parameters and sends requests to the skeleton on the server side. The skeleton, in turn, extracts arguments, invokes the actual method on the remote object, captures the result, and marshals it back to the stub. This interaction ensures the seamless execution of remote methods and the exchange of data between the client and server components.

iii) Transport Layer:

The transport layer plays a pivotal role in controlling the references called by clients for remote objects within the RMI architecture. It handles the transmission of data between components, ensuring the integrity and efficiency of communication.

iv) Remote Reference Layer (RRL):

In charge of establishing and maintaining connections between the client and server, the Remote Reference Layer (RRL) is an important component in the RMI architecture which oversees all existing and currently active connections, ensuring a robust and reliable linkage between the client and server for seamless remote method invocations.

Data Flow and Communication Protocols in RMI

A white paper with black text

Description automatically generated

Figure 4: Data Flow in RMI

The above figure depicts the data flow process visualization, which visualized the flowing phases of the communication between the client and server components in a Java RMI application environment..

JRMP (Java Remote Method Protocol):

The Java Remote Method Protocol (JRMP) is used in the context Java RMI for secure transmission of data between Java Virtual Machines. JRMP is combination of rules that control the serialized transmission of data between clients and servers. This communication protocol enables the smooth and continuous invocation of methods on remote objects, making sure that a continuous network connection within the RMI framework. Java RMI achieves a standardized and secure communication route by conforming to JRMP, which is required for the reliable operation of distributed applications.

# Implementation

In the online shopping application development, the following are the implementations done from my side in our group project. The purchase or ordering part of Items was done by me where I have executed the principles of an RMI environment as explained below:

## 4.1 RMI Application Development Process with Code Snippets

An RMI application system is developed by the following process:

1. Definition of Remote Interface

The first thing done was to define the remote interfaces required for communication between the client and the server in the RMI application.

A computer screen shot of text

Description automatically generated

Figure 6: Defining Remote Interface

Here, interface is created where certain methods are declared that are executed in purchase part of this RMI application.

1. Development of remote object

After the definition of remote interfaces, next this to be done is implementing the remote interfaces to create remote object. This Java interface includes method declarations for remote invocation by clients.

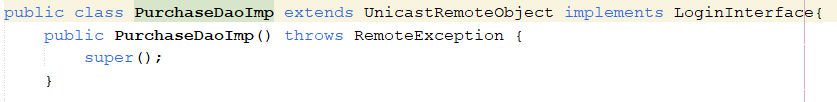


Figure 7: Developing Remote Object and its Interface

A computer screen shot of a code

Description automatically generated

Figure 8: RMI Method Declaration I

A screen shot of a computer code

Description automatically generated

Figure 9: RMI Method Declaration II

A screenshot of a computer code

Description automatically generated

Figure 10: RMI Method Declaration III

A screen shot of a computer

Description automatically generated

Figure 11: RMI Method Declaration IV

A screen shot of a computer code

Description automatically generated

1. Developing the client program

A client-side program is created once the remote object and interface have been established. This software interfaces the client with the remote object, allowing users to create and interact with it.

A screen shot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

Figure 12: Client program

1. Compiling the source files and client stubs and server skeletons generation:

The next step is to compile Java source files to build skeletons and stubs for both the remote object class and the client software. This is accomplished by utilizing the javac or Java compiler, which produces class files.

1. Starting the RMI Registry:

The RMI registry was created to allow users to look for remote objects by name. It's in the same folder as the classes.

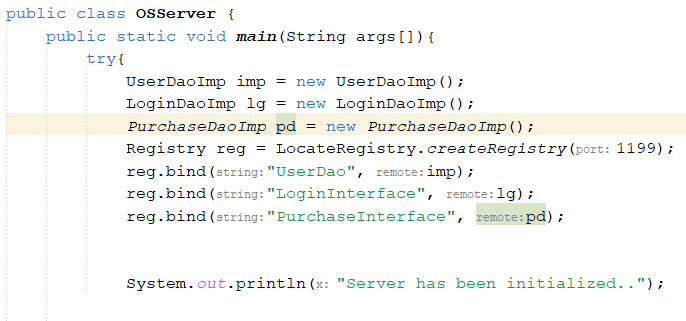


Figure 13: RMI Registry

1. Starting the remote object and running the client:

The server is launched after the registry has been started, allowing it to hold server information. The client software is run with the necessary configurations. Implementing the RMI registry to look for remote objects and enable method invocation is required.

A screenshot of a computer program

Description automatically generated

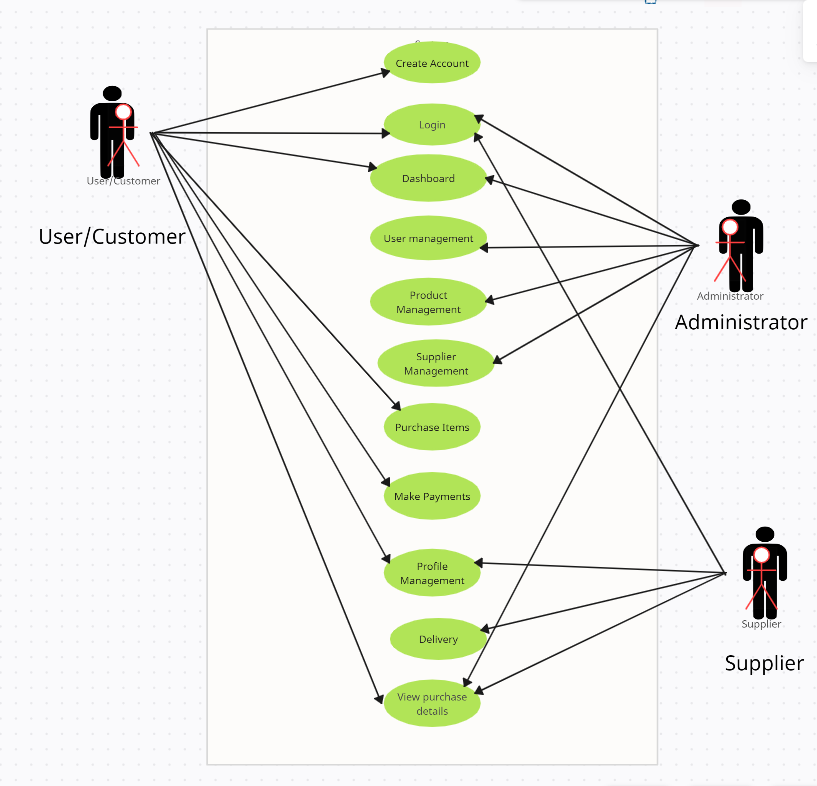
Figure 14: Server Setup

A screenshot of a computer code

Description automatically generated

Figure 15: Client Lookup

## 4.2 Use Case Diagram



The use case diagram in figure shows the interactions between a user, a supplier, and an administrator in the context of a Online Product Delivery System. The system allows users to create accounts, login, view product details, purchase items, make payments, manage their profiles, and view purchase details. The administrator can manage users, products, suppliers, and deliveries. The use case diagram provides a high-level overview of the system and how it is used by the different actors. It can be used to communicate the system's requirements to stakeholders and to guide the development of the system.

# Testing

In order to ensure the development of this application was done properly, certain testing functions were carried out by my team. Since I have implemented the purchasing or the ordering part of the application I have performed certain tests regarding this part of application. Here, proper brief regarding the testing of my component is given below:

## Integration Testing

As there were different components for each member in our team, I have chosen integration testing as an method of testing my component as it will ensure tha specific area of the application is good to integrate with the main application. This testing provided mw with brief results of tests performed in the purchasing area of the application. In the following table brief results of the test Is provided.

This testing ensures that users can effectively search for products, view accurate product details, select and add products to their cart, and proceed to checkout seamlessly. It also verifies that the cart summary reflects the selected products and their pricing, available payment options are displayed, and users can complete the payment process and track their order delivery status.

Deployment in the Cloud:

Deploying the RMI application in a cloud computing environment entail deciding on a cloud service provider, such as AWS or Azure, and then deciding on the suitable architecture, such as Virtual Machines or containers. The use of the cloud improves worldwide accessibility, scalability, and flexibility. Configuring networks, implementing load balancing, and ensuring security are all critical stages. Potential network latency and security concerns must be addressed as challenges. Overall, cloud deployment improves resource utilization and enables dynamic scaling in response to application demand.

Benefits and Challenges of Cloud-Based Deployment:

Scalability, flexibility, cost effectiveness, and worldwide accessibility are all advantages of cloud-based deployment. Organizations can reduce costs by using a pay-as-you-go model, and cloud services are available globally. However, difficulties include network latency and security concerns, particularly when storing and sending data. A potential issue is vendor lock-in, underlining the significance of implementing cloud-agnostic methods. Regulation compliance is critical, and careful consideration of these elements assures effective cloud adoption.

Future Enhancements:

Exploring developing cloud computing and virtualization technologies and trends can pave the way for future improvements. Integration of serverless computing, virtualization improvements, and support for AI and IoT are all promising areas of investigation. Furthermore, the application could benefit from blockchain for increased security. By adhering to these trends, the RMI application may remain inventive and flexible to changing technology landscapes.

Conclusion

Finally, the RMI application development project successfully addresses CKF's Product Delivery System issues. Using RMI and deploying on the cloud improves efficiency and scalability. The system's reliability is validated by testing methodologies, and future developments are in line with emerging technology. The project lays the groundwork for future developments in cloud computing and virtualization, ultimately achieving its goals of improving the delivery system and staying current with emerging technology trends.